errors of the received messages, and to provide parity for transmitted messages.

A block diagram of an A-Net port 198, particularly a downstream port, is shown in Fig. 3. A state machine 200 receives the RTS*, CTS and CLK signals to aid in the handshaking with the other port. A parity generator/checker 202 is connected to the DATA < 7..0 > and PAR signals. When receiving, the parity generator/checker 202 checks the parity of the received data and transmits an error from its parity error (PE) output to the state machine 200 for synchronization and transmission to the IOC control logic. On transmissions the parity generator/checker 202 develops the parity value and drives the PAR signal. A receive latch 204 is connected to the parity generator/checker 202 to latch received data before transmission to the receive message buffer associated with the
ANP. Command decode/encode logic 206 is connected to the output of the receive latch 204 to allow decode of the command byte, with the outputs of the decode/encode logic 206 being provided to the state machine 200 to allow cycle length tracking and to the IOC control logic, and to the inputs of an transmit latch 208 to allow command output. The transmit latch 208 receives data from the transmit message buffer associated with the ANP and the decode/encode logic 206 and provides it to the parity generator/checker 202. The state machine 200, through signals not shown, controls the data input and output by the latches 204 and 208, the direction of the parity generator/checker 202 and the decode/encode timing of the command decode/encode logic 206.

An IOC 230, as shown in Fig. 4, includes control logic 232, a set of local registers 236; an upstream port 234 and multiple downstream A-Net ports 238. For a simple IOC 230, the upstream port 234 and all of the downstream ports of IOCs are
A-Net ports and comprehend the channel protocol.

The functions of the IOC control logic are to decode the address of a received message from any port to decide whether the message is for the on-board registers of the
IOC or for others nodes located below the IOC, to queue commands from all ports, to schedule received commands from all ports and control the various transmit and receive buffers, to encode messages in case of exception conditions, and to synchronize varying transfer rates at the downstream ports.

An exemplary organization of an IOC 230 is in Figure 4, based on bus transfer between the upstream and downstream ports, but other configurations, such as a multiplexed/demultiplexed structure, crosspoint organization or others could also be used. The upstream port 234 has associated with it a receive message buffer 240 and a transmit message buffer 242. The receive message buffer 240 is connected to the control logic 232 to allow for the address decode of the message. The message buffers 240 and 242 are
FIFOs. The upstream port 234 is connected to the control logic to allow error, command and other values to be communicated.

The output of the receive message buffer 240 is provided to a receive bus 244. The input of the transmit message buffer 242 is connected to a transmit bus 246.

The various downstream ports 238 are coupled to the receive and transmit buses 244 and 246. Each downstream port 238 has associated transmit and receive message buffers 248 and 250, respectively. The input of the transmit buffer 248 is connected to the receive bus 244, while the output of the receive buffer 250 is connected to the transmit bus 246. The receive message buffer 250 and the downstream port 238 are both connected to the control logic 232 for like reasons as the upstream port 234. Only the firstd Nthdownstream -ports. 23-8 are shown in Fig. 4, but it is understood that all of the ports are identical.

A transfer gate 252 has its inputs connected to the transmit bus 246 and its outputs connected to the receive bus 244 to allow peer-to-peer transmission between the various devices downstream of the IOC 230. The local registers 236 have their inputs connected to the receive bus 244 and their outputs connected to the transmit bus 246. Not shown in Figure 4 for clarity are the various control lines from the control logic 232 to latch and output enable the various buffers, registers, and gates. Also not shown in Figure 4 for clarity is the clocking logic used to receive and generate the necessary clock signals to advance state machines and shift registers and synchronize latching and output enabling of the various devices. It is understood that the upstream and downstream ports may have different basic clock frequencies.

The illustrated IOC uses FIFO's for each port, but only a single set of FIFO's is required in a minimal configuration.

The MIOC 120 does similar functions as the IOC 230 except that the upstream port of the MIOC 120 is designed to interface with the host bus 106 and except for the following functions.

The MIOC 120 decodes the messages from the downstream devices and converts them to respond to the processor or to access the system memory 112, encodes the memory responses into response messages to the requested devices, decodes host bus 106 accesses to respond to A-Net addressing as well as to be coherent with the CPU caches, encodes the processor requests into messages for the devices, and interrupts the processor when errors are reported from the downstream devices.

The FIFOs for each port may be relatively short, such as the length of the longest message, or can be much longer to act as prefetch and write posting buffers. The IOC can monitor the ReadS and WriteS Commands. When a ReadS Command is received, the IOC, especially the MIOC, can obtain additional information from sequential locations before the next Read Command is received. This prefetching reduces arbitration sequences, allowing a performance increase. It is especially helpful when the block size of the requested device is larger than the A-Net packet block size. A new ReadS Command need not be issued upstream by the prefetching unit until the prefetched data has been transferred. Similarly, write posting can be performed using the WriteS command and a longer FIFO.Several sequential packets can be stored in the FIFO and the entire chain is not transmitted until the FIFO is full or a non-WriteS Command is received. This write posting also reduces the number of arbitrations necessary, increasing overall system performance.

The IOC control logic 232 contains the logic necessary to implement and control the prefetching and write posting.

Preferably one additional feature can be performed by the IOC control logic 232. As each channel within an IOC is arbitrated, it is desirable to balance the devices across the various IOCs. This balancing can be performed during development for permanently attached A-Net devices or periodically for slotted A-Net devices. To facilitate this balancing, performance monitoring capabilities can be included in each IOC and device. An exemplary way this can be done is to provide a counter for each channel. The counter increments on backoffs or retries on the channel, either upstream or downstream. Alternatively, the counters can count the length of the waiting periods of transfers on the channels. The counters can be cleared by a request from the controlling processor.Operations would be performed for a desired sampling period and then the controlling processor would read the various counters. The controlling processor would then review the counts and suggest optimal locations for the various devices. To reduce possible channel latency problems affecting the counter values, the data can be provided out-of-band over a serial link. Preferably two highly used devices would not be installed on one IOC while another IOC at the same level had none. Further, this would also allow movement of devices among tree levels to balance loading at each level.

An IOD is used to interface a peripheral device or a subsystem, such as a SCSI to the A-Net, or used to drive an externally located A-Net subsystem via a non-A-Net interface.

An IOD only has an upstream A-Net port with associated ANP logic and register set, and clock logic if necessary. The downstream portion corresponds to the interface needed for the particular peripheral device. The IOD generally has circuitry to perform the desired peripheral function. The IOD control logic preferably is a command list processor and retrieves the command list that processor created, encodes messages to send upstream to "execute" the command-list, and processes the received messages. As noted above, both the IOC 230 and the IODs have registers. Certain registers are used for operation of the protocol and these are described below.

Figure 5 illustrates the local registers in the IOC 230.

The registers local address and operation are as follows.

A-Net~Device~Type: address 000h
This 32-bit register contains a unique device type indicator used during the boot process (described below) to determine the topography of the current A-Net implementation and its associated devices. The bits define the presence of Option ROMs, 3 character manufacture code, 3 hex digit product number, and a 1 hex digit product

revision number as follows:
Byte 3
Bit 7 Option ROM (Present if Set)
Bit 6:2 Character 1
Bit 1:0 Character 2 (upper bits)
Byte 2
Bit 7:5 Character 2 (lower bits)
Bit 4:0 Character 3
Byte 1
Bit 7:4 Product Digit 1
Bit 3:0 Product Digit 2
Byte0
Bit 7:4 Product Digit 3
Bit 3:0 Revision Digit
Supported characters are ASCII values 41h (A) through 5Ah (Z), and compression is achieved by discarding the three most significant bits of a byte. Thus A is 00001b and Z is 11010.

Remaining digits are in standard hexadecimal representation.

IOC Address Mask: address 008h
This 8-bit register is set at configuration/boot time. It is used to support geographical addressing of the IOC hierarchy. EachIOC's address mask will be set based on the system topology to allow it to determine if later cycles are addressed to it or to devices or IOCs connected to one of its channels.

IOC~Configuration: addressO0Ch
This 32-bit read only register supplies the boot/configuration information about the particular implementation of this IOC and includes, for example, the number of downstream channels and their activity status.

IOC Exception Status: address 010h
This 32-bit register reflects the exception or error condition that caused the IOC to perform an operation to the
IOC~Exception Address.

IOC~Exception~Address: address 018h
This register is encoded at configuration/boot time by the appropriate processor to indicate the address of the controlling processor's programmable interrupt controller (PIC) and interrupt level under error or exception conditions. This register has a length as indicated by the Address Size command.

IOC Channel Status: Base address 020h (Channel 1)
This 8-bit register contains values representing the status of the particular IOC channel.

Bit 3 Enable/Disable Channel (Enabled if Set
default)
Bit 2:0 Transfer Clock Rate
0 - > 25MHz
1 - > 50MHz
2 - > 100MHz
3 - > 200MHz IOC~Channel~TimeOut: Base address 022h (Channel 1)
This 8-bit register contains integer values representing the number of contiguous idle clocks before an IOC will stop the clock of a given channel.

TheIOC channel status and IOC~channel~timeout registers for the remaining channels follow successively, until registers have been provided for all the channels in the IOC.

Figure 6 illustrates the local registers in anIOD used for A-Net communications. The name, address and operation are as follows:
A-Net~Device~Type: address 000h
This 32-bit register contains a unique device type indicator used during the boot process to determine the topography of the current A-Net implementation and its associated devices.Bits 31 to 0 define the presence of Option

ROMs, 3 character manufacture code, 3 hex digit product number, and a 1 hex digit product revision number as follows:
Byte 3
Bit 7 Option ROM (Present if Set)
Bit 6:2 Character 1
Bit 1:0 Character 2 (upper bits)
Byte 2
Bit 7:5 Character 2 (lower bits)
Bit 4:0 Character 3
Byte 1
Bit 7:4 Product Digit 1
Bit 3:0 Product Digit 2
Byte0
Bit 7:4 Product Digit 3
Bit 3:0 Revision Digit
Supported characters are ASCII values 41h (A) through 5Ah (Z), and compression is achieved by discarding the three most significant bits. Thus A is0000lib and Z is 11010. Remaining digits are in standard hexadecimal representation.

Dev~Start~Port: address 008h
This register is written by the processor which has just enqueued a command packet for the device in system memory. The register has the length indicated by the Address Size command.

Note that the simple act of writing the port with any value will minimally cause whatever series of events is required to cause the device to seek out the command list in main memory.

In implementations with Option ROMs at boot time, this value must be the address where the initial4K block is to be transferred. In other implementations, the device designer and device driver designer may implement extensions using the data value. Conventionally, writing the value will cause the device to issue a command list read request at that address.

Dev Interrupt Address: address 010h
A-Net devices that may cause interrupts back to the system bus implement this register. The Interrupt~Address~Port is encoded at configuration/boot time by the appropriate processor to indicate the address of the controlling processor's PIC and interrupt level. This register has the length indicated in the Address Size command.

Dev~Exception~Address: address 018h
A-Net devices that wish to report an error or exception condition separate from the normal interrupt back to the host bus 106 implement this register which is encoded at configuration/boot time by the appropriate processor to indicate the address of the controlling processor's PIC and interrupt level under error or exception conditions. This register has the length indicated in the Address Size command.

Dev~Interrupt~Port : address 020h
This port receives a standard A-Net write which will cause an interrupt to the designated device. Note that no specific value must be written to the port. The simple act of writing the port with any value will minimally interrupt the device.

In addition to the variousIOC's andIOD's having registers, the PIC 108 is also memory mapped and contains a number of registers containing interrupt vectors or levels.

Each level is a separate memory address, so that the PIC 108 then occupies a range of memory space. The values in the
Exception~Addresses and the Interrupt~Address correspond to these locations.

Dev Status Port : address 028h
This 32-bit read/write register contains the following information:
Bit 31:12 Reserved
Bit 11 Gross Error (if set device is in an error state)
Bit 10 Endian (Clear if Little Endian default)
Bit 9 Boot Mode (if set device enters boot mode)

Bit 8 Reset (if set device enters reset state)
Bit 7:3 Reserved
Bit 2:0 Transfer Clock Rate
0 - > 25 MHz
1 - > 50 MHz
2 - > 100 MHz
3 - > 200 MHz
Programmable Interrupt Controller(PIC): base address
XXXXXXXXh
The PIC 108 decodes the address range starting at address
XXXXXXXXh and extending to equal the number of possible interrupt levels to addresses. For example, the address range for 256 interrupt levels would be 00000000h to000000FFh, for a base address of 00000000h.The PIC 108 implements an operating system settable register for each of the addressable interrupt levels. The register value is returned to the processor during the interrupt acknowledge sequence as the interrupt vector. This allows the operating system to modify the hierarchy of interrupt vectors from the processor's point of view.

The A-Net architecture encompasses three forms of addressing: downstream, upstream, and peer to peer. Each address type utilizes a geographical addressing scheme allowing for maximum flexibility. Each IOC or device is guaranteed to have at least its own 4k addressable region. The general format for geographical addresses that are used during the addressing of A-Net devices is shown below.
EMI36.1

<tb>

n <SEP> (n-w) <SEP> x <SEP> O
<tb> System <SEP> Mosk/Address <SEP> #dres# <SEP>
<tb>
The (n-w) most significant bits (MSbs) of the address are system dependant. The x least significant bits (LSbs) are guaranteed address bits with a minimum size of 12 bits, thus giving each IOC and device its minimum 4k page of memory. The remaining (n-(w+x)) mask/address bits serve multiple purposes based on the type of addressing and will be described further below. To illustrate addressing, the address format shown will implemented with n = 40, w = 16, and x = 12.

Downstream addressing is defined as addresses (and associated other information) flowing from the host bus 106 to the MIOC 120 and down through the levels of IOCs 140 and 146 to reach a device. Certain conventions cover addressing. The
MIOC 120 has the ability to recognize host bus 106 cycles intended for the A-Net structure, and is a master on the host bus 106. When an IOC other than the MIOC 120 sees an address, it is for the IOC or one of its channels. When a device sees an address, the address is for it. Each IOC has up to2"-1 channels, with the downstream channels numbered 1 to 2n-1.

Logical channel 0 is defined as theIOC's local address space.

For purposes of address decoding, address Byte/Bit
Numbering is as follows:
Byte no.:0 1 2 3 4
Bit no. : 0123 4567 0123 4567 0123 4567 0123 4567 0123 4567
The 8-bit IOC Address Mask register of each IOC has the following format:
xxxNNbbb
Where: N is the byte number and b is the bit number. The number of bits N defined in this register corresponds to the number of address bytes available to address a device. The number of bits in the "b" field corresponds to the number ofdevices'attached to an IOC including theIOC itself. Three bits allow up to 7 devices to attach to an IOC. Seven devices is considered a preferable limit as that keeps the pin count of the IOC chip at a cost effective level.Bits 4 and 3 indicate the byte, numbered from Most Significant to Least Significant, i.e. bits 39 - 32 represent transfer byte 0. Bits 2 through 0 indicate the Most Significant bit of the mask field, with the bits numbered in reverse order, bit 7 = 0 and so on.

During the boot/configuration of the system, the system boot master processor sets theIOC Address Mask for each IOC in the A-Net configuration. From that time forward the appropriate portion of each

downstream address is used to determine if the address is for the IOC itself, the defined mask bits in address = 0, or one of its channels, the defined mask address bits being greater than 0. In the event that the value is not in the range of channels, the IOC will generate an
Exception to the appropriate processor as specified in theIOC~Exception~Address register with the appropriate bits of theIOC~Exception~Status register set.For the sample system shown in Figures 1A and 1B, theIOC~Address~Masks would be set as follows:
MIOC 120 - > XXX01000 (bits 23-21, start: byte 1, bit 0)IOC 140 - > XXX01011 (bits 20-19, start: byte 1, bit 3)IOC - > XXX01011 (bits 20-19, start: byte 1, bit 3)
IOC 146 - > XXX01101 (bits 18-17, start: byte 1, bit 5)
Note that IOC 140 and the next listedIOC, which is provided only for example and is not shown in Figures1A and 1B but is assumed to be connected to the MIOC 120, have the same value in theIOC~Address~Mask register. This does not present a problem as each IOC will only see addresses intended for it or its children, by definition.Also, the bits of the mask/address field that are not used for the geographical map in the parent
IOC may be used by the downstream devices as address bits.

As an address enters the MIOC 120, it is checked and passed on to the appropriate channel. For instance, an address with bits 23-21 set to 3 would cause the address to be relayed to IOC 140. Via this method, the address finally arrives.

downstream at the device for which it is intended, the device automatically assuming that .the address. is intended for it.

Upstream addressing is defined as a device releasing an address that moves through the A-Net hierarchy and out to the system bus. In this case, all the mask/address bits may be used as address bits in addition to the normal 12 bits, allowing the maximum addressable page on the system bus.

Peer-to-peer addressing comprises both upstream and downstream addressing, and is defined as an A-Net device performing a write operation to another A-Net device without the intervention of the host bus 106. It is assumed that the device is aware of the geographical or downstream address that it must release to reach the appropriate destination. EachIOC must be able to interpret this incoming 'upstream' address and determine if it should be the IOC to turn the packet 'downstream'. This is facilitated by aIOC Top Loc orIOC topography location register which contains the topographical location of the given IOC.The IOC will compare the incoming 'upstream' address with theIOC Top Register, and if the mask information indicates that the incoming packet is for theIOC or one of its channels, will turn the packet back downstream.

Using a 40-bit address, a system can use a maximum of 28 bits to address a device and the remaining 12 bits to address the memory within the device. The most significant 28 bits of the address essentially route data packets through the IOC tree to a specific I/O device and the 12 remaining bits address memory within that device. A device may require more than 12 bits of address space to fully access its memory. In this case less than 28 bits are used to route the data to the device and the remaining bits are used to address the device's memory.

The A-Net structure lends itself well to the use of geographical addressing to access the major components, such as the memory 112, the graphics controller 114, the MIOC 120 and so on. For this discussion assume that the most significant address byte defines the system geographical address. Assume that the MIOC 120 resides at the geographical address defined by byte 0 above being 15h. Assume also that the MIOC's 120IOC Address Mask register contains a value of 08h (0000 1000b).

From the address mask register template above, bits NN areOI and bits bbb are 000. This means that the MIOC 120 will use
Byte 1, bits0,1 1 and 2 tobegin its address decode.

The decoding scheme defines the host IOC as device 0 and the remaining devices on the IOC links as devices 1 to2n - 1, where n is the number of bits "b". For this example the MIOC 120 registers reside in the address space beginning 0001 0101 000x xxxxb in bytes 0 and 1 respectively. Device 1 of the MIOC 120 has its address space beginning at 0001 0101 001x xxxxb.

This continues until device 7, which has the same byte 0 value, 0001 0101b, but byte 1 equals lllx xxxxb.

The MIOC 120 and its devices use the remaining 29 bits to address memory in their own address spaces.

This example continues with IOC 140 attached to the MIOC's first link. This corresponds to device 2 and will receive all data packets whose address bits begin with 0001 0101 010x xxxx...xxxxb. To take advantage of its entire address range, the IOC 140 will begin its decode at byte 1, bits 3, 4 and 5.

TheIOC's address mask register therefore contains 0000 1011b.

This defines the IOC 140 local address space, as device 0, to begin at 0001 0101 010000xx.. .xxxxb. The first device's address space begins at 0001 0101 0100..... xxxxb and so on until the seventh device. Thus the IOCs closer to the MIOC 120 control larger address spaces, while the IOCs farther from the root control smaller address spaces.

An A-Net transfer can be initiated by a processor or any
A-Net node. The processor reads from or writes to devices during system initialization and system management, such as, interrupts due to error conditions. Most processor initiated transfers are small packets, usually to the Start~Port, for example, to direct to a command list. The majority of the routine A-Net transfers are initiated by the MIOC 120 and the devices. Intermediate-level IOCs can also initiate transfers in case of error conditions.

For processor device transfers, the processor informs a device which transfer types are to be initiated via a commandlist. A command-list consists of device specific commands.

After a command-list is set up by the processor, it notifies the corresponding device to read the command-list. The device reads the command-list, fully or partially depending on the device implementation, into a local command list buffer for processing.

A device can initiate a transfer to read/write the memory 112 or another device. The device to memory transfers are best described by using examples. The following paragraphs demonstrate how a device local command list is executed to transfer data to and from the memory 112. It is assumed that there are no errors during these transfers.

Assume that IOD-1 has only one upstream port and the local command list of IOD-1 consists of the following commands
Entry-11: Device-Read ... (read data from the peripheral device)
Entry-12: Write to00FFFE0000h, 2048 bytes
Entry-13: Read from 00FFF20000h, 4100 bytes
Entry-14: Device-Write ... (write data to the peripheral device)
For all A-Net transfers, the transfer width for each port (and channel) is 8-bits and the maximum data length of a packet is 32 bytes.The number of A-Net packets required to complete the
Entry-12 and Entry-13 commands are
Total #
Packets of Bytes# of A-Net
Entry-12: 2048 64 32-byte packets
Entry-13: 4100 128 32-byte packets + 1 4
byte packet
For Entry-12,IOD-1 encodes 64 packets with each packet containing the Write command, the size, the address and the associated 32-byte data:
Packet# Size Address
1 3200FFFE0000
2 3200FFFE0020
3 32 00FFFE0040
64 32 00FFFE07E0
These write messages will be received by the MIOC 112 and theassociated data will be written to.thememory by the.MIOC 112.

For Entry-13,IOD-1 encodes 129 packets, with each packet containing the Read command, the size and the address;
Packet# Size Address
1 32 00FFF20000
2 32 00FFF20020

128 32 00FFF20FE0
129 4 00FFF21000
After the MIOC 120 receives the read messages from IOD-1, it will read the memory starting from the address00FFF20000. The
MIOC 120 will then encode read response messages along with the read data and send them back to IOD-I. Since each IOD-1 read request received by the intermediate IOCs is satisfied with an
MIOC 120 read response message in order, the read response messages do not need addresses. IOD-I will eventually receive 64 read response messages, each with 32 data bytes and one read response message with 4 data bytes.

Preferably all device-to-device or peer-to-peer transfers between two devices are via write messages. The processor is responsible to build the command list for the devices, so that the device that has the data will write to the requesting device. All transfers are strongly ordered within an IOC, both requests/responses from/to a port and requests/responses from/to the ports.

The A-Net system supports a fully symmetrical multiprocessing interrupt model for system master to system master, system master to A-Net, and A-Net to system master interrupts.

This scheme allows system masters to interrupt each other, a system master to interrupt a device on the A-Net, or an A-Net device to interrupt the appropriate processor. The interrupt vector implementation is flexible to allow system software designers to implement hierarchial or linear, or both, interrupt levels at their discretion.

The PIC 108 interfaces to the host bus 106 and decodes all interrupts intended for its processor, as described above; determines if it should interrupt the processor with the current. pending interrupt; supports the interrupt function to the processor and supplies the interrupt acknowledge interface to the processor.

The interface to the host bus 106 is primarily implementation dependant. Minimally, the PIC 108 must be able to decode addresses released onto the host bus 106 to determine if it is the intended recipient. Note that the host bus 106 cycle may be a read, write or exchange. The device initiating the sequence will not expect valid data to be returned, but uses the read and exchange operations as synchronization tools.

The PIC 108 decodes the address range starting at an implementation specific base address and extending to equal the number of possible interrupt levels to addresses, as noted above. The PIC 108 sends the highest ranked pending interrupt to the processor during the interrupt acknowledge cycle(s).

Ranking is determined by address, with the base address representing the lowest priority interrupt. For instance, if a level 15 interrupt causes the PIC 108 to assert the interrupt line to the processor, and a level 23 interrupt arrives before the processor begins the interrupt acknowledge sequence, the level 23 vector will be written to the processor. The PIC 108 implements an operating system settable register for each of the addressable interrupt levels, with the register value to be returned to the processor during the interrupt acknowledge sequence.

Each A-Net device, including IOCs, that requires an incoming interrupt implements a Device~Interrupt~Port at local address 020h, as described above. This port receives a standard A-Net write, which causes an interrupt to the designated device. Two possible schemes to remove the interrupt condition are a hardware oriented interrupt acknowledge to the local port or a return interrupt to the appropriate system master causing a clearing operation to the port.

A-Net devices that may cause interrupts back to the host bus 106 implement a port to support this operation. The
Interrupt~Address~Port as described above provides this function. All IOCs and all devices that wish to report an error or exception condition separate from the normal interrupt back to the host bus 106 implementa port to support. this operation. The Exception~Address register as described above provides this function.

If a system master wishes to interrupt another, system master, all the interrupting system master need do is perform a read operation to the other System Master's local PIC address. For a system master to interrupt an A-Net device, the system master generates a write to the appropriate address for the given device and the Device~Interrupt~Port in the device's local address space. This will cause an interrupt to

the device. For an A-Net device to interrupt a system master, the
A-Net device simply performs a read using the address supplied in the Interrupt~Address register.

Moving now to the various cycle types that make up the A
Net signaling protocol, all A-Net ports sample their inputs on the falling clock edge. Upstream ports drive their signals on rising edges. Downstream ports drive on falling edges. A port will not drive any signal due to any input until one cycle after that input is sampled.

The timing diagrams are Figures 7A, 7B, and 8-13 and represent 50 MHz operation. In the timing diagrams the letter "C" represents command byte, the letter "A" represents address bytes and the letter "D" represents data bytes. The "C", "A", and "D" are prefixed by a letter "U" for an upstream port sending or a letter "D" for a downstream sending, e.g., "UA" means an address byte sent by the upstream port. Device and
IOC state indications are explained by the state machines of
Appendix A, which show idle, transmit and receive states and actions for devices and IOCs. Some of the figures show upstream and downstream signals in separate diagrams along with some phase shift to illustrate how phase shifting affects when signals reach the ports. All signals are shown driven with a delay after the appropriate edge to represent clock-to-output delay.The dotted lines in the figures align with falling edges. Dashed lines align with rising edges where confusion might otherwise exist.

Referring now to Figures 7A and 7B, in a single message transfer, a sender port samples its CTS signal TRUE on a falling edge of CLK (-3) and asserts its RTS* signal during the next cycle (-2). The sender also samples it's CTS signal. at
CLK (-2) and drives the data lines starting with the command byte during the next cycle (-1). The sender continues to sample its CTS signal and drive the remainder of the bytes on each clock. The sender samples its CTS signal FALSE at the falling edge of CLK (2). This is an acknowledgement by the receiver that it recognizes the incoming message. The sender must sample its CTS signal TRUE at the remaining CLKs at the remaining falling edges to follow the standard single cycle.

If a sender ever samples its CTS signal FALSE then another cycle type will be followed. The sender finally drives the last byte of the message and at the same time de-asserts its
RTS* signal to complete the transfer.

Referring now to Figure 8, the receiver port receives two consecutive messages from the sender port in a back-to-back transfer. If the CTS signal is sampled TRUE when the last byte is sent (state TX) the sender is allowed to send back-to-back messages. The sender de-asserts it's RTS* signal as the last byte of the message is sent, CLK (4) above, as in a standard single cycle. The sender immediately reasserts it's RTS* signal, CLK (5), and another single message transfer begins.

Sampling the CTS signal FALSE on the CLK (4) does not void the current message but simply precludes a back-to-back transfer.

If a message is shorter than the one illustrated, the time between messages extends appropriately. Longer messages simply have repeating TX and RX states. Note that the receiver could assert it's RTS* signal during the last transfer states (R3 or
RX). If this occurs, the protocol enters a collision sequence and resolves according to those rules.

Referring now to Figure 9, if a port has a message ready to be sent and that port is currently receiving a message, it may request the current sender to reverse the channel at the end of the current message. The receiving port asserts it's
RTS* signal at any time during the transmission after CLK 3, but is only guaranteed a reversal if it gets to the sender in time for the sender to sample it on the CLK (-1) that the last byte is sent. The sender thus knows, by sampling it's CTS signal (-1), that it is unable to send a back-to-back message (if it had desired) and that it can expect an incoming message.

The. receiver may sendit's command byte if it already has asserted it's RTS* signal and if it samples the CTS signal high. Note that a channel reversal requires a dead clock. If a message is shorter than the one illustrated, the time between messages extends appropriately.

If the receiver asserts it's RTS* signal too late, it is possible that the transmission might collide with the sender's back to back transmission. If this occurs, the normal collision resolution takes place. If the

receiving port knows how long the incoming message is because it decoded the command byte (and the size field if necessary) then it may use this information to avoid the assertion of RTS* signal at the end of the senders message (and subsequent possible collision). This will prevent collisions, but would increase the time to the start of the next message if the sender did not run a back to back cycle. The first set of signals in the figures show what would occur if the channel had no phase shift. The separate upstream and downstream signal sets show a view with phase shift.

Referring now to Figure 10, a sender port is backed-off when the receiver port on the other end of the channel currently cannot completely receive the current message. The receiver can backoff the sender by keeping its RTS* signal asserted on the clock following the acknowledgement so that the sender will sample its CTS signal FALSE at state T3. The sender will backoff and de-assert its RTS* signal at the next clock (3 or state TB3). The receiver will discard any part of the message already received.

If the receiver does not want to send a message then it will de-assert its RTS* signal on the next clock, CLK (3). The sender should then retry sending the message (or another higher priority message) as shown.

Referring now to Figure 11, if the receiver wants to send a message it will continue to assert its RTS* signal and send bytes starting with the first clock after it samples its CTS signal TRUE. The figures' first set of signals show what would occur if the channel had no phase shift. The separate upstream and downstream signal sets show a view with phase shift. Note that the first message in each of the figures is terminated early because of the backoffand retry. Note that. to backoff the sender, the receiver must assert its RTS* signal on CLK (2) so that the sender will sample CTS signal FALSE on the falling edge of CLK (3).

Referring now to Figure 12, Channel Matched Collision occurs when both A-Net ports on a channel issue their RTS* signal on the same clock cycle as in the first case [-2] in
Figure 12. Both ports sample their CTS signals FALSE during arbitration (stateTM1). The upstream port always wins and will continue to assert its RTS* signal and hold its first byte valid for an extra clock (stateTMlx) and then proceed normally. The loser (downstream port) de-asserts it's RTS* signal before the second CLK when it realizes the collision (see firstcase).

If a downstream port asserts its RTS* signal during the first byte of a transmission by the upstream port, an Early
Collision has occurred (see second case). In an Early
Collision both RTS* signals are asserted in arbitration but data does not actually collide. This will happen if the downstream port tries to start a message one cycle later than the upstream port. The downstream port will realize at state
TM2 that it cannot send but does not have the time to keep from asserting its RTS* signal. The downstream port will go into receive mode and de-assert its RTS* signal (state R0). The sender (upstream port) simply ignores its CTS input during the first byte cycle.

If an upstream port asserts its RTS* signal during the first byte of a transmission by the downstream port, a Late
Collision has occurred (see third case). In a Late Collision both RTS* signals are asserted in arbitration but data does not actually collide. This will happen if an upstream port tries to start a message one cycle later than the downstream port.

The upstream port will realize at state TM2 that it cannot send but does not have the time to keep from asserting its RTS* signal. The upstream port will go into receive mode and deassert its RTS* signal (state R0). The sender (downstream port) simply ignores its CTS input during the first byte cycle.

As in any computer system, the various devices must be configured after reset and the system booted. The goals of the
A-Net configuration/boot processinclude determining the topology of the subsystem, initializing all IOCs, determining all devices in the system and taking appropriate action for each, locating all bootable devices, presenting the loaded operating system with all necessary information about the A-Net subsystem and booting the operating system from the appropriate device.

The following paragraphs illustrate the steps required to configure and boot an A-Net subsystem, with Figures 13A and 13B showing portions in a flowchart format.

The following boot sequence is started at the MIOC 120 and iterated throughout the configuration of an A-Net subsystem.

First, read theIOC's IOC~Config register to determine the number of channels the IOC supports. Next, set theIOC's IOC Address Mask to reflect the number of channels. Then generate a read of the A-Net Device~Type register for each channel in turn. Based on the device, perform appropriate action. This four step process is continued until the entire topology of the particular A-Net implementation is determined, and all devices have been identified.

The many different device types that are returned by theA-Net~Device#Type register read may be grouped into three basic categories based on the following characteristics. First, a device that the configuration code recognizes. Second, a device that the configuration code does not recognize. Third, a device that is another IOC. If the device is one that the configuration code recognizes, the code simply performs any necessary initialization and returns to the channel check iteration explained above. In the event that the device is not recognized, minimally the configuration code informs the operating system at boot time about the device's type identifier and its location in the A-Net topology. If the device is another IOC, the IOC initialization steps outlined above are simply executed for the next level IOC.

If the queried channel does indeed support a device, it may certain an option ROM and/or be bootable. If bit 31 of the
A-Net~Device~Type value returned by the device is SET, the device has an option ROM that must be initialized. Upon recognizing that the device has an option ROM, the system master initializes the device'sDeyInterrupt~Address- and performs a write to the device's Dev~Start~Port. The value of the write will be the location to which the device should move its first4K of code. When the device has completed the4K code transfer, it generates a cycle to the address placed in the Dev~Interrupt~Address. After recognizing the interrupt, the system master will begin execution of the ROM code at the address written to the Dev~Start~Port.The device's option ROM code should notify the configuration code of its device type via aDevice~RON~Type service and perform any necessary initialization of the device and return control to the configuration code.

Note that in the event that the device is bootable, it must also call the configuration code serviceBoot~Device#Entry to inform the configuration code of the location and size of the device's boot code as well as the boot priority of the device.

Bootable devices are recognized in one of two ways.

First, the boot code recognizes the A-Net~Device~Type value read from the device as one that is bootable. Secondly, the device may have option ROMs that inform the configuration code of the fact that it is bootable. The configuration code determines all bootable devices in the A-Net system during the determination of the topology and passes this information to the system boot code. To determine the primary bootable device, the system boot code chooses the highest priority option ROM based bootable device to attempt boot. Should that fail, any remaining Option ROM based bootable devices will be tried in the order of their priority. Finally, each system ROM understood device will be tried in the order of their implementation dependant priorities.

The information on how to boot a device will be found in one of two places. In the case of a device that is recognized by the configuration code, the system boot code will have the knowledge required to boot the device. If option ROMs are used, the call to the Boot~Device~Entry system service will include the device's boot entry.

The hot plugging of devices is a special sub-set of the normal A-Net boot procedure. The effected IOC must generate an exception to the appropriat#e system master, asdefined in the IOC Exception Address register, both when an A-Net expansion board is removed and/or when one is inserted. The exception handler will then read theIOC Exception Status register, which will inform the handler that a device was removed from the system or a new device was 'hot plugged' into the given channel. The syste#m master will immediately perform the boot procedure outlined above on the indicated channel, taking appropriate action as necessary.

Proceeding now to Figures 13A and 13B, the IOC initialization operations are shown in a flowchart format. The computer system resets and commences the reset sequence 300.

At step 302 various initialization operations are performed.

Conventionally this would include testing portions of the memory 112 to assure that the RAM is satisfactory, performing any necessary testing of the cache 104 and the boot processor 100 and other certain minimal and immediate initialization operations. After these initial operations are concluded, control proceeds to step 304 where a structure is set up to initialize the MIOC 120. Control then proceeds to step 306 where the IOC INIT sequence 350 is called. After the IOC INIT sequence 350 is finally completed and the entire chain has been initialized and determined, control proceeds to step 308 where the computer system C continues the booting process as described above.

The IOC INIT sequence 350 commences at step 352 where the
IOC configuration register is read. This determines how many channels are located on the downstream side of the IOC and which of these channels has a device connected to it. Control then proceeds to step 354 where theIOC~Address~Mask is set and the Interrupt Address is loaded. Control proceeds to step 356 where an IOC channel counter is set to 1. Control proceeds to step 358 where the A-Net device type register for the particular channel count is read to determine what device is attached to that particular port. If the device is unknown, control proceeds to step 364 where a flag is set to indicate to the configuration code the unknown status. Control then proceeds to step 362.If the next level device is anIOC, control proceeds to step 366 where structures are set up for a next levelof IOC. Control then proceeds to step368 where the
IOC INIT sequence 350 is again called. Therefore it is clear that the IOC INIT sequence 350 must be reenterable so that these threads can begin. After the IOC INIT sequence is completed in step 358, control proceeds to step 362.

If it is determined in step 360 that the device type was either a known type or indicated an option ROM, control proceeds to step 370 to determine if an option ROM was present.

If not, control proceeds to step 372 where the proper initialization operations for the particular device are performed. Control then proceeds to step 362.

If an option ROM was present as determined in step 370, control proceeds to step 374 where the Interrupt~Address andStart~Port registers are written for that particular device.

Writing to the Start~Port device as described above causes the device to transfer the first4K of the option ROM to the address provided. This is shown in the dotted operation step 376. While step 376 and the following step 378 are asynchronous events and performed in parallel, they are shown in this flowchart to show actual data flow operation. In step 376 the device transfers the ROM code to the starting address at the Start~Port and issues an interrupt back-up to the boot processor. The boot processor is interrupted and executes step 378 which is an execution of the ROM code which has been transferred to memory. This code includes placing the device type in the proper table as noted above and initializing the device. Control then proceeds to step 362.

In step 362 a determination is made as to whether this is the last channel for this particular IOC. If not, control proceeds to step 380 where the IOC channel counter is incremented to the next active channel and control proceeds to step 358 to reiterate for the next channel. If this was the last channel, control proceeds to step 382 which is a return from the sequence.

The foregoing disclosure and description of the invention are illustrative and explanatory thereof, and various changes in the size, shape, materials, components, circuit elements, wiring connections and contacts, as well as in the details of the illustrated circuitry and construction and method of operation may be made without departing from the spirit of the invention.

APPENDIX A 50 MHzDevice State Machine (Idle and Transmit States)
EMI53.1

<tb> State <SEP> Evert <SEP> Next <SEP> Action <SEP> Comment <SEP>
<tb> <SEP> late <SEP>
<tb> Midle <SEP> Idle <SEP>
<tb>Idle <SEP> <SEP> RD <SEP> <SEP> Receive <SEP> incoming <SEP>

<tb> <SEP> CTS-SEND <SEP> <SEP> Tm2 <SEP> <SEP> <SEP> RTS <SEP> Begin <SEP> Transmit <SEP>
<tb> <SEP> CTS*/SEND <SEP> Idle <SEP> Idle <SEP>
<tb> TM2 <SEP> CTS <SEP> <SEP> TM1 <SEP> <SEP> Drive <SEP> command <SEP>
<tb> <SEP> /CTS <SEP> RD <SEP> <SEP> float <SEP> RTS <SEP> late <SEP> collision <SEP>
<tb>ml <SEP> CTS <SEP> TO <SEP> Drive <SEP> data
<tb> <SEP> /CTS <SEP> RM1 <SEP> float <SEP> RTS, <SEP> data <SEP> matched <SEP> collision
<tb> To <SEP> <SEP> Drive <SEP> data <SEP>
<tb>Ti <SEP> <SEP> T2 <SEP> Drive <SEP> data <SEP>
<tb> T2 <SEP> CTS <SEP> <SEP> Error
<tb> <SEP> /CTS <SEP> T3
<tb> T3 <SEP> CTS <SEP> <SEP> TX <SEP> Drive <SEP> <SEP> Data <SEP> No <SEP> Boll <SEP>
<tb> <SEP> /CTS <SEP> TB4 <SEP> float <SEP> data, <SEP> RTS <SEP> <SEP> BOFFed
<tb> TB4 <SEP> <SEP> TM2 <SEP> <SEP> Drive <SEP> <SEP> RTS <SEP> Retry <SEP>
<tb> <SEP> /CTS <SEP> <SEP> Midle <SEP> Full <SEP> Backoff <SEP>
<tb> TX <SEP> /Done <SEP> TX <SEP> Drive <SEP> <SEP> data
<tb> <SEP> CTS*Done*Send <SEP> TXB <SEP> Drive <SEP> <SEP> RTS <SEP> Back <SEP> to <SEP> back <SEP>
<tb> <SEP> CTS*Done*/Send <SEP> Midle <SEP> Idle
<tb> <SEP> /CTS*Done <SEP> Midle <SEP> Reverse <SEP>
<tb> <SEP> /RTS-/CTS <SEP> <SEP> Idle <SEP> Quick <SEP> <SEP> Reverse <SEP>
<tb> TXB <SEP> CTS <SEP> <SEP> <SEP> Drive <SEP> command <SEP>
<tb> <SEP> /CTS <SEP> Idle <SEP> float <SEP> RTS <SEP> B2B <SEP> collision
<tb> Table-1 50 MHz device idle and transmit state table.

50Mhz Device State Machine; Receive States
EMI54.1

<tb> State <SEP> Event <SEP> Next <SEP> Action <SEP> Comment <SEP>
<tb> <SEP> state <SEP>
<tb> R0 <SEP> pantyok <SEP> <SEP> R1 <SEP> assert <SEP> RTS <SEP>
<tb> <SEP> parity <SEP> bad <SEP> <SEP> Error <SEP>
<tb> RRX <SEP> Send <SEP> TM1 <SEP> <SEP> Drive <SEP> Command <SEP> <SEP> Begin <SEP> transmission
<tb> <SEP> /Send <SEP> Idle <SEP> <SEP> Walt <SEP> for
<tb> R1 <SEP> backoff <SEP> R2 <SEP> assert <SEP> RTS <SEP> need <SEP> to <SEP> backoff <SEP>
<tb> <SEP> ok <SEP> <SEP> R2 <SEP> float <SEP> RTS
<tb> R2 <SEP> Send <SEP> R3 <SEP> continue <SEP> RTS <SEP> full <SEP> backoff <SEP> or <SEP> reversal <SEP>
<tb> <SEP> /Send <SEP> R3 <SEP> Coal <SEP> <SEP> RTS <SEP> retry <SEP> or <SEP> no <SEP> reversal <SEP>
<tb>R3 <SEP> Send~ <SEP> R4 <SEP> assert <SEP> RTS <SEP> try <SEP> for <SEP> reversal <SEP>
<tb> <SEP> /backoff <SEP>
<tb> <SEP> else
<tb> R4 <SEP> backoff <SEP> RRX <SEP> retry <SEP> or <SEP> full <SEP> beckoff <SEP>
<tb> <SEP> Send <SEP> <SEP> RX <SEP> <SEP> assert <SEP> RTS <SEP> try <SEP> for <SEP> reversal <SEP>
<tb> <SEP> ok*/Send <SEP> RX <SEP>
<tb> <SEP> ok*CTS-RTS <SEP> <SEP> TM1 <SEP> <SEP> Drive <SEP> Command <SEP> Quick <SEP> Reversal <SEP>
<tb> RX <SEP> <SEP> /CTS*/RTS*Send <SEP> RX <SEP> <SEP> assert <SEP> RTS <SEP> try <SEP> for <SEP> reversal <SEP>
<tb> <SEP> /CTS*/RTS*/Send <SEP> RX <SEP> <SEP> continue <SEP> receive <SEP>
<tb> <SEP> CTS*/RTS*Send <SEP> TM2 <SEP> assert <SEP> RTS <SEP> try <SEP> to <SEP> send <SEP>
<tb> <SEP> CTS-/RTS-/Send <SEP> <SEP> Idle <SEP> <SEP> idle
<tb> <SEP> CTS*RTS <SEP> TM1 <SEP> drive <SEP> <SEP> command <SEP> <SEP> begin <SEP> transmit

<tb> <SEP> /CrS*RTS <SEP> RX <SEP> reverse, <SEP> wait <SEP> for <SEP> rx <SEP> done
<tb>
Table-2 50 Mhz device receive state table.

50 MHz IOC State Machine (Idle and Transmit States)
The states are named for the dock after thefalling edge where the inputs are firstsampled. the outputs will begin to take effect starting on the falling clock edge at the endot the state (tor devices, 1/2 docklater tor IOC).
EMI55.1

<tb>

SIte ENent Next Action comment
Mate
Midle Idle
Idle /CTS R0 Receive incoming
CTS*SEND TM2 assert RTS Begin Transmlt
CTS*/SEND Idie Idle
m2 CTS TM1 DykeDrive command
/CTS R0 float RTS late collision
TM1 CTS T0 Drive data
/CTS Tm1x Drive command again matched collision
TM1X TO Drive data
TO <tb>T1 <SEP> <SEP> T2 <SEP> Drive <SEP> data
<tb> T2 <SEP> CTS <SEP> Error
<tb> <SEP> /CTS <SEP> T3
<tb> T3 <SEP> CTS <SEP> TX <SEP> Driven <SEP> data <SEP> no <SEP> boff <SEP>
<tb> <SEP> /CTS <SEP> TB4 <SEP> floatdata <SEP> RTS <SEP> <SEP> BOFFED
<tb> TB4 <SEP> CTS <SEP> TM2 <SEP> Drive <SEP> RTS <SEP> Retry
<tb> <SEP> /CTS <SEP> Midle <SEP> Full <SEP> Packet <SEP>
<tb> TX <SEP> /Done <SEP> TX <SEP> Drive <SEP> data <SEP>
<tb> <SEP> CTS*Done* <SEP> TXB <SEP> Drive <SEP> RTS <SEP> <SEP> Back <SEP> to <SEP> back <SEP>
<tb> <SEP> Send
<tb> <SEP> CTS-Done*/Send <SEP> <SEP> Midle <SEP> idle
<tb> <SEP> /CTS*Done <SEP> Midle <SEP> Reverse
<tb> <SEP> /RTS*/CTS <SEP> We <SEP> Quick <SEP> Reverse
<tb> TXB <SEP> CTS <SEP> TM1 <SEP> Drive <SEP> command <SEP>
<tb> <SEP> /CTS <SEP> Idle <SEP> <SEP> float <SEP> RTS <SEP> 828 <SEP> collision <SEP>
<tb>
Table-3 50 MHz IOC idle and transmit state table.

50 MHz IOC State Machine; Receive States
EMI56.1

State Event Next Action
Mite
RD parity R1 assert RTS
parity bad Error
RRX Send TM1 Drive Command Begin Transmission
/Send We Wait tor receive
R1 backoff R2 assert RTS need to backoff
ok Ra float RTS
R2 Send R3 <tb> <SEP> /Send <SEP> <SEP> retry <SEP> or <SEP> no <SEP> reversal <SEP>
<tb>R3 <SEP> <SEP> Send*/backoff <SEP> R4 <SEP> assert <SEP> RTS <SEP> try <SEP> for <SEP> reversal <SEP>
<tb> <SEP> else <SEP> R4
<tb> R4 <SEP> backoff <SEP> RRX <SEP> retry <SEP> or <SEP> full <SEP> backoff <SEP>
<tb> <SEP> ok*Send*/RTS <SEP> , <SEP> RX <SEP> assert <SEP> RTS <SEP> try <SEP> for <SEP>

reversal <SEP>
<tb> <SEP> oiP/Send <SEP> <SEP> RX <SEP>
<tb> <SEP> ok*CTS*RTS <SEP> TM1 <SEP> Drive <SEP> Command <SEP> <SEP> Quick <SEP>
Reversal
<tb>RX <SEP> <SEP> /CTS*/RTS*Send <SEP> RX <SEP> <SEP> absen <SEP> RTS <SEP> <SEP>
try <SEP> for <SEP> reversal <SEP>
<tb> <SEP> /CTS*/RTS*/Send <SEP> RX <SEP> continue <SEP> <SEP> receive
<tb> <SEP> CTS*/RTS*Send <SEP> TM2 <SEP> assert <SEP> RTS <SEP> try <SEP> to <SEP> send
<tb> <SEP> CTS*/RTS*/Send <SEP> Idle <SEP> Idle <SEP>
<tb> <SEP> CTS*RTS <SEP> TM1 <SEP> any. <SEP> <SEP> <SEP> begin <SEP> tranmit <SEP>
<tb> <SEP> /CTS*RTS <SEP> RX <SEP> <SEP> reverse,wait <SEP> for <SEP> rx <SEP> done <SEP>

<tb>
Table4 50 MhzIOC receive state table.

SCALABLE TREE STRUCTURED HIGH SPEED I/O SUBSYSTEM ARCHITECTURE

Claims of **WO9414121**

CLAIMS:

1 1. An input/output subsystem for a computer system
2 having a means to provide and receiving addresses and data to
3 perform operations, the input/output subsystem comprising:
4 a plurality of addressable input/output devices
5 receiving and providing packets, each of said packets having a
6 plurality of fields according to a predetermined protocol, said
7 packets including a command portion and being formed of a
8 plurality of elements; and
9 means connected to said means for providing and 10 receiving addresses and data and each of said plurality of 11 input/output devices for transferring data between said means 12 for providing and receiving addresses and data and each of said 13 plurality of input/output devices, each of said plurality of 14 input/output devices connected to said means for providing and 15 receiving addresses and data separately from each other of said 16 plurality of input/output devices, 17 wherein said means for transferring includes: 18 means for determining if an operation or a 19 plurality of operations is directed to one of said plurality of 20 input/output devices by analyzing said address portion of said 21 operation; 22 means for converting said address and data 23 operation or said plurality of address and data operations to 24 a packet; 25 means for transmitting said packet 26 developed from said address and data operation or said 27 plurality of address and data operations to said one of said 28 plurality of input/output devices as a series of said plurality 29 of elements forming said packet; 30 means for receiving a packet from one of 31 said plurality of input/output devices as a series of said 32 plurality of elements forming said packet; 33 means for converting said packet received 34 from said one of said plurality of input/output devices to an 35 address and data operation or a plurality of address and data 36 operations; and 37 means for providing said address and data 38 operation or said plurality of address and data operations to 39 said means for providing and receiving addresses and data.

1 2. The input/output subsystem of claim 1, wherein each
2 of said plurality of input/output devices includes a port for
3 receiving said packet from said means for transferring and for
4 transmitting said packet to said means for transferring, and
5 wherein said means for transmitting said packet and
6 said means for receiving said packet comprise a plurality of
7 ports for transmitting said packet and receiving said packet,
8 with each of said plurality of ports connected to a respective
9 port of one of said plurality of input/output devices.

1 3. The input/output subsystem of claim 2, wherein said
2 means for transmitting and said means for receiving include a
3 first level device having a first number of ports and at least
4 one concentrator device, each said concentrator device having
5 a port connected to one of said first level ports or a
6 concentrator device and further having a predetermined number
7 of said ports for connection to input/output devices or further
8 concentrator devices, wherein said first number of ports of
9 said first level device and said total number of predetermined 10 ports of said concentrator devices less the number of said 11 concentrator devices is greater than or equal to the number of 12 said plurality of input/output devices.

1 4. The input/output subsystem of claim 3, wherein read
2 operations are performed as split transactions and each of said
3 concentrator devices includes a queue of port numbers for
4 maintaining strict ordering of read operation responses.

1 5. The input/output subsystem of claim 3, wherein a
2 concentrator device includes means for prefetching read
3 operation data and for posting write operation data for one
4 port.

1 6. The input/output subsystem of claim 3, wherein said
2. first level device,each said concentrator device and each of 3 said plurality of input/output devices includes a plurality of 4 addressable registers, wherein each of said plurality of 5 addressable registers includes a register indicating the type 6 of device.

1 7. The input/output subsystem of claim 6, wherein said 2 plurality of addressable registers in said input/output devices 3 further includes a register for receiving an address to which 4 to send an interrupt, a register at which to receive an 5 interrupt request and a register at which to receive a memory
6 address indicating a location of a data structure.

1 8. The input/output subsystem of claim 6, wherein said 2 plurality of addressable registers of said first level device
3 further includes a register indicating the value of said first 4 number of ports, said plurality of addressable registers if 5 each said concentrator device further includes a register
6 indicating the value of said predetermined number, and said 7 plurality of addressable registers in said first level device
8 and each said concentrator device further includes a register
9 indicating an address decode location for determining if an 10 address provided on said bus is directed to said first level 11 device or said concentrator device or one of said ports of said 12 first level device or said concentrator device.

1 9. The input/output subsystem of claim 8, wherein the
2 address used to address an input/output device comprises a
3 plurality of bytes and wherein the value in said address decode
4 location register indicates the byte and the bit starting in
5 the byte used for address decode to determine if said packet is
6 for said first level device, said concentrator device or to a
7 device coupled to one of said ports.

1 10. The input/output subsystem of claim 2, wherein an
2 input/output device includes two of said ports and two of said
3 ports of said means for transmitting said packet and said means
4 for receiving said packet are connected to said two ports on
5 said input/output device.

1 11. The input/output subsystem of claim 2, wherein said
2 data is transmitted in a parallel manner between the means for
3 providing and receiving addresses and data and the input/output
4 subsystem and between said ports, and wherein said means for
5 transmitting and said means for receiving further include:
6 means for receiving said packet from said means for
7 transferring in a parallel data format and for transmitting
8 said packet to said means for transferring in a parallel data
9 format and for transmitting said packets received from said 10 means for transmitting in a serial data format and for 11 receiving packets for transmitting to said means for receiving 12 in a serial data format; and 13 means for receiving a packet in serial data format from 14 said means for transferring in a serial data format and 15 transmitting said packet from a port in a parallel data format 16 and for transmitting a packet in a serial data format to said 17 means for transferring in a serial data format received at said 18 port in a parallel data format.

1 12. The input/output subsystem of claim 11, wherein the
2 computer system includes two separate housings and one of said
3 parallel to serial and serial to parallel means is located in
4 one of said housings and the other of said parallel to serial
5 and serial to parallel means is located in the other of said
6 housings.

1 13. The input/output subsystem of claim 1, wherein said
2 command portion of said packet indicates a data movement
3 command or a control or status command.

1 14. The input/output subsystem of claim 13, wherein said
2 data movement commands include read requests, write requests,
3 exchanges, compare and swaps and read responses.

1 15. The input/output subsystem of claim 14, wherein said
2 read and write request commands include indications of the
3 number of data bytes to be transferred.

1 16. The input/output subsystem of claim 14, wherein said 2 read and write requests and said exchange and said compare and 3 swap include the starting address of the data.

1 17. The input/output subsystem of claim 13, wherein said 2 control or status commands include exception indications.

1 18. A computer system comprising: 2 a plurality of processors, each processor including 3 a central processing unit performing address and data 4 operations, memory and an input/output system interface means 5 to provide and receive packets to perform operations, each of
6 said packets having a plurality of fields according to a 7 predetermined protocol, said packets including a command
8 portion and being formed of a plurality of elements and being
9 directed to a particular address; 10 a plurality of addressable input/output devices 11 receiving and providing packets, each of said input/output 12 devices including means connected to said input/output 13 interface means of each of said processors for providing and 14 receiving packets, each of said plurality of input/output 15 devices connected to said input/output interface means of each 16. of said processors separately from each other of said plurality 17 of input/output devices, 18 wherein each of said input/output system interface 19 means includes: 20 means for determining if an operation or a 21 plurality of operations is directed to one of said plurality of 22 input/output devices by analyzing said addresses provided by 23 said central processing unit; 24 means for transmitting said packet developed 25 from said address and data operations or said plurality of 26 address and data operations to said one of said plurality of 27 input/output devices as a series of said plurality of elements 28 forming said packet; 29 means for receiving a packet from one of said 30 plurality of input/output devices as a series of said plurality 31 of elements forming said packet; and 32 means for converting said packet received from 33 said one of said plurality of input/output devices to an 34 address and data operation or a plurality of address and data 35 operations.

1 19. The computer system of claim 18, wherein each of said
2 plurality of input/output devices includes a port for receiving
3 said packet from each of said input/output system interface
4 means of each of said processors; and
5 wherein said input/output system interface means of
6 each of said processors includes a plurality of ports for
7 transmitting said packet and receiving said packet, with each
8 of said plurality of ports connected to a respective port of
9 one of said plurality of input/output devices.

1 20. The computer system of claim 19, wherein an
2 input/output device includes two of said ports and two ports of
3 said input/output system interface means are connected to said
4 two ports on said input/output device.

1 21. A computer system comprising:
2 a microprocessor performing address and data
3 operations and having a graphics interface means to receive and
4 provide packets and an input/output system interface means to
5 receive and provide packets, each of said packets having a
6 plurality of fields according to a predetermined protocol, said

7 packets including a command portion and being formed of a
8 plurality of elements and being directed to a particular
9 address; 10 a main memory connected to said microprocessor; 11 a graphics controller connected to said graphics 12 interface means; and 13 an input/output system connected to said input/output 14 system interface means, said input/output system including: 15 a plurality of addressable input/output devices 16 receiving and providing packets; and 17 means connected to said input/output system 18 interface means and each of said plurality of input/output19 devices for transferring data between said microprocessor and 20 each of said plurality of input/output devices, each of said 21 plurality of input/output devices connected to said means for 22 transferring data separately from each other of said plurality 23 of input/output devices, wherein said means for transferring 24 includes: 25 means for determining if a packet is 26 directed to one of said plurality of input/output devices by 27 analyzing said address portion of said packet; 28 means for transmitting said packet to said 29 one of said plurality of input/output devices as a series of 30 said plurality of elements forming said packet; 31 means for receiving a packet from one of 32 said plurality of input/output devices as a series of said 33 plurality of elements forming said packet; and 34 means for providing said received packet to 35 said input/output system interface means.

1 22. The computer system of claim 21, wherein each of said
2 plurality of input/output devices includes a port for receiving
3 said packet from said means for transferring and for
4 transmitting said packet to said means for transferring, and
5 wherein said means for transmitting said packet and
6 said means for receiving said packet comprise a plurality of
7 ports for transmitting said packet and receiving said packet,
8 with each of said plurality of ports connected to a respective
9 port of one of said plurality of input/output devices.

1 23. The computer system of claim 22, wherein said
2 input/output system interface means includes a port for
3 transmitting said packet and receiving said packet and wherein
4 said means for transmitting and said means for receiving
5 includes at least one concentrator device, each said
6 concentrator device having a port connected to said
7 input/output system interface means or a concentrator device
8 and further having a predetermined number of said ports for
9 connection to input/output devices or further concentrator 10 devices, wherein said total number of predetermined ports of 11 said concentrator devices less the number of said concentrator 12 devices is greater than or equal to the number of said 13 plurality of input/output devices.

1 24. The computer system of claim 23, wherein each said
2 concentrator device and each of said plurality of input/output
3 devices includes a plurality of addressable registers, wherein 4 each of said plurality of addressable registers includes a
5 register indicating the type of device.

1 25. The computer system of claim 24, wherein said
2 plurality of addressable registers in said input/output devices
3 further includes a register for receiving an address to which
4 to send an interrupt, a register at which to receive an
5 interrupt request and a register at which to receive a memory
6 address indicating a location of a command list.

1 26. The computer system of claim 24, wherein said
2 plurality of addressable registers of each said concentrator
3 device further includes a register indicating the value of said
4 predetermined number, and each said concentrator device further
5 includes a register indicating an address decode location for
6 determining if an address provided with said packet is directed
7 to said concentrator device or one of said ports of said
8 concentrator device.

1 27. The computer system of claim 26, wherein the address 2 used to address an input/output device

comprises a plurality of 3 bytes and wherein the value in said address decode location 4 register indicates the byte and the bit starting in the byte 5 used for address decode to determine if said packet is for said 6 first level device, said concentrator device or to a device 7 coupled to one of said ports.

1 28. A computer system comprising: 2 a bus having address, data and control portions, said 3 data portion having a first width; 4 a processor having address, data and control signals, 5 said signals being connected to said bus;
6 a main memory connected to said bus; and
7 an input/output subsystem connected to said bus, said
8 input/output subsystem including:
9 a plurality of addressable input/output devices 10 receiving and providing packets, each of said packets having a 11 plurality of fields according to a predetermined protocol, said 12 packets including a command portion and being formed of a 13 plurality of elements having a width less than said first width 14 of said data portion of said bus; and 15 means connected to said bus and each of said 16 plurality of input/output devices for transferring data between 17 said bus and each of said plurality of input/output devices, 18 each of said plurality of input/output devices connected to 19 said means for transferring data separately from each other of 20 said plurality of input/output devices, wherein said means for 21 transferring includes: 22 means for determining if a bus operation or 23 a plurality of bus operations is directed to one of said 24 plurality of input/output devices by analyzing said address 25 portion of said bus; 26 means for converting said bus operation or 27 said plurality of bus operations to a packet; 28 means for transmitting said packet 29 developed from said bus operation or said plurality of bus 30 operations to said one of said plurality of input/output 31 devices as a series of said plurality of elements forming said 32 packet; 33 means for receiving a packet from one of 34 said plurality of input/output devices as a series of said 35 plurality of elements forming said packet; 36 means for converting said packet received 37 from said one of said plurality of input/output devices to a 38 bus operation or a plurality of bus operations; and 39 means for providing said bus operation or 40 said plurality of bus operations to said bus.

1 29. The computer system of claim 28, wherein each of said
2 plurality of input/output devices includes a port for receiving
3 said packet from said means for transferring and for
4 transmitting said packet to said means for transferring, and
5 wherein said means for transmitting said packet and
6 said means for receiving said packet comprise a plurality f 7 ports for transmitting said packet and receiving said packet,
8 with each of said plurality of ports connected to a respective
9 port of one of said plurality of input/output devices.

1 30. The computer system of claim 29, wherein said means
2 for transmitting and said means for receiving includes a first
3 level device having a first number of ports and at least one 4 concentrator device, each said concentrator device having a
5 port connected to one of said first level ports or a
6 concentrator device and further having a predetermined number
7 of said ports for connection to input/output devices or further
8 concentrator devices, wherein said first number of ports of
9 said first level device and said total number of predetermined 10 ports of said concentrator devices less the number of said 11 concentrator devices is greater than or equal to the number of 12 said plurality of input/output devices.

1 31. The computer system of claim 30, wherein said first
2 level device, each said concentrator device and each of said
3 plurality of input/output devices includes a plurality of
4 addressable registers, wherein each of said plurality of
5 addressable registers includes a register indicating the type
6 of device.

1 32. The computer system of claim 31, wherein said
2 plurality of addressable registers in said input/output devices
3 further includes a register for receiving an address to which
4 to send an interrupt, a register at which to receive an
5 interrupt request and a register at which to receive a memory

6 address indicating a location of a command list.

1 33. The computer system of claim 31, wherein said
2 plurality of addressable registers of said first level device
3 further includes a register indicating the value of said first
4 number of ports, said plurality of addressable registers if
5 each said concentrator device further includes a register
6 indicating. the value of said predetermined. number, and. said
7 plurality of addressable registers in said first level device
8 and each said concentrator device further includes a register
9 indicating an address decode location for determining if an 10 address provided on said bus is directed to said first level 11 device or said concentrator device or one of said ports of said 12 first level device or said concentrator device.

1 34. The computer system of claim 33, wherein the address
2 used to address an input/output device comprises a plurality of
3 bytes and wherein the value in said address decode location
4 register indicates the byte and the bit starting in the byte
5 used for address decode to determine if said packet is for said
6 first level device, said concentrator device or to a device
7 coupled to one of said ports.

1 35. The computer system of claim 28, wherein said command
2 portion of said packet indicates a data movement command or a
3 control or status command.

1 36. The computer system of claim 35, wherein said data
2 movement commands include read requests, write requests,
3 exchanges and read responses.

1 37. The computer system of claim 36, wherein said read 2 and write request commands include indications of the number of
3 data bytes to be transferred.

1 38. The computer system of claim 37, wherein said data
2 movement commands include a command o allow an increase of the
3 number of data bytes to be transferred by a read or write
4 request command.

1 39. The computer system of claim 36 wherein said read and
2 write requests and said exchange include the starting address
3 of the data.

1 40. The computer system of claim 35, wherein said control
2 or status commands include exception indications.